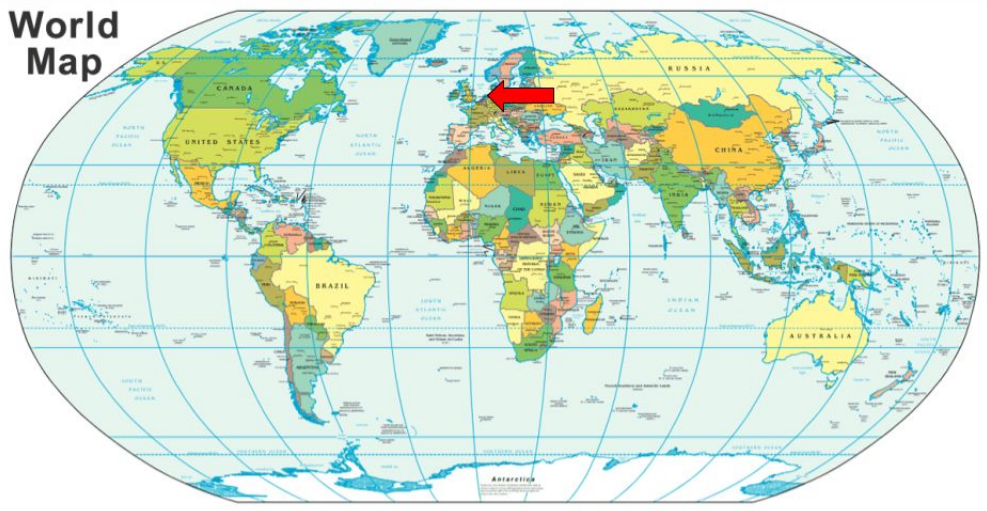
How NPO achieved High-Throughput, Secure Video Streaming Running NGINX on Modest Off-The-Shelf Hardware

Dick Snippe - Dutch Public Broadcasting Organization (NPO)

Overview

- Geography of the Netherlands
- Load Balancing
- Content protection using Lua
- SSL Optimizations

Geography of the Netherlands

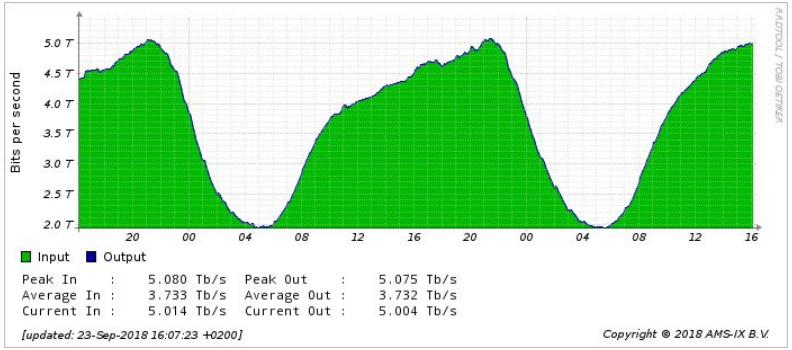


Geography of the Netherlands

- It's small; 41.543 km² ≈16.000 mi² About Massachusetts + Connecticut; not even 1% of the USA land area
- But quite densely populated; 17 million inhabitants; that's like cramming all of Texas into West Virginia
- It's all flat. No mountains. Easy to lay cables.
- Ideal for good internet connectivity.
- 98% of households has broadband internet
- 4G coverage is about 98%
- Amsterdam is the main internet hub

Internet topology

- ams-ix is the main hub.
- two large cable providers
- place your gear in Amsterdam and you're all set!



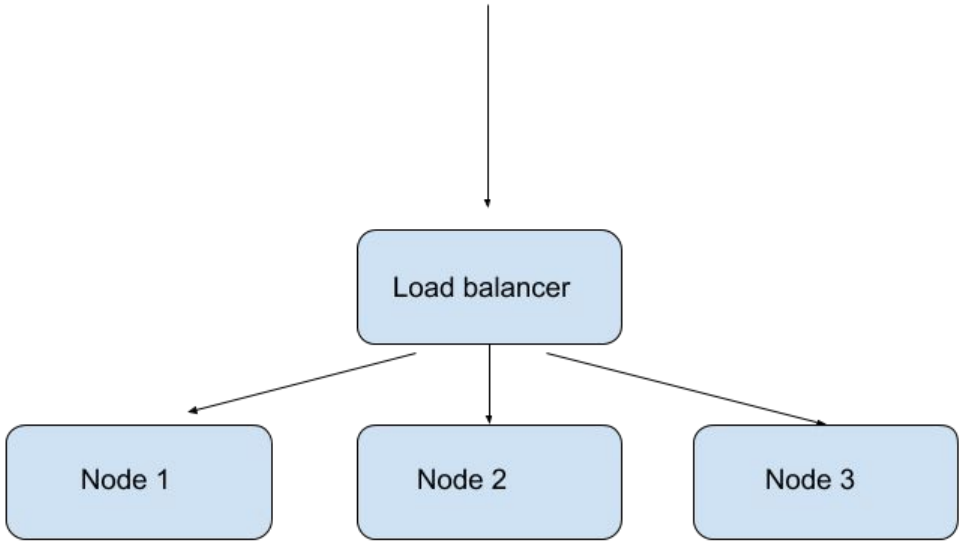
Loadbalancing

Loadbalancing

Classical loadbalacing: 1 (or a HA pair) loadbalancer; all traffic (upstream+downstream) passes through the loadbalancer

This is fine for most websites / api's

Loadbalancing



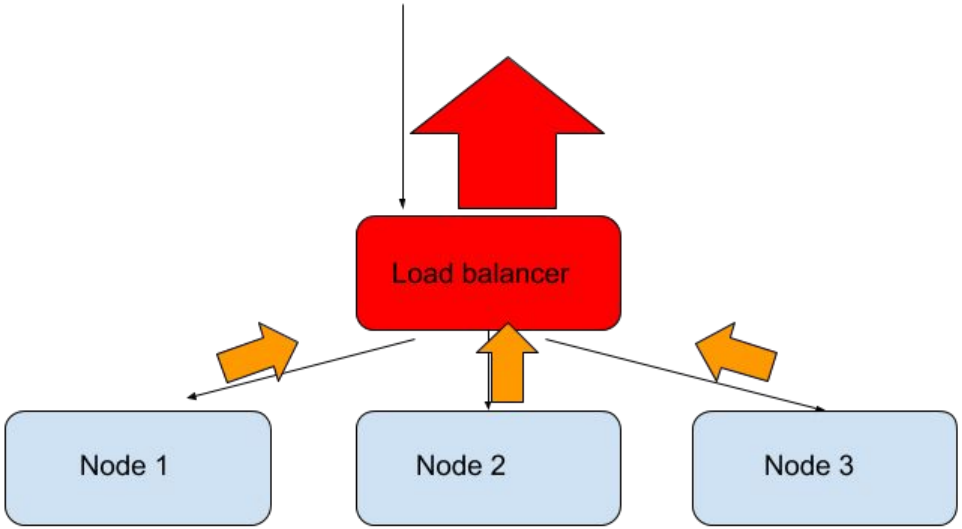
Loadbalancing

Classical loadbalacing: 1 (or a HA pair) loadbalancer; all traffic (upstream+downstream) passes through the loadbalancer

This is fine for most websites / api's

Not so good for streaming where upstream traffic is 1000x the downstream traffic. The loadbalancer becomes the choke point when is has to handle >10Gbit of traffic

Loadbalancing



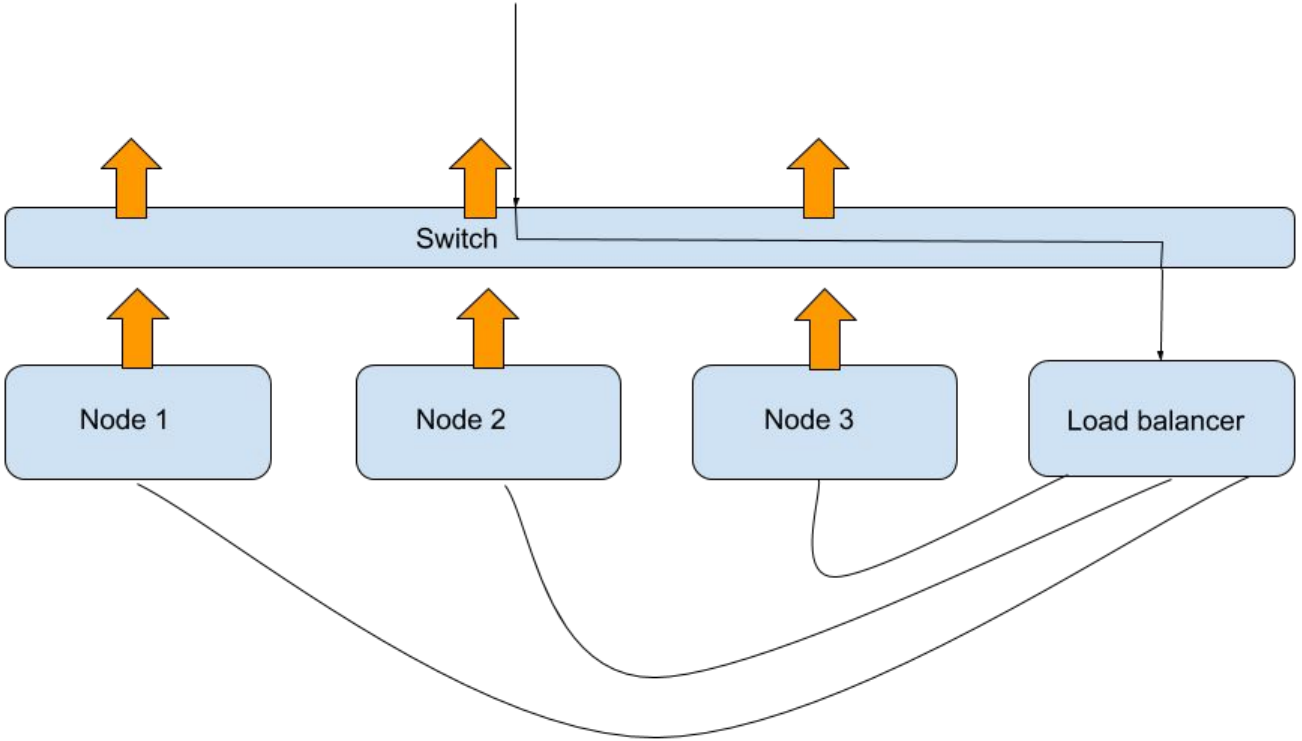
Loadbalancing

A nice approach is "Direct Routing"; IPVS-DR. A linux kernel module that acts as a loadbalancer.

Only the downstream traffic flows through the loadbalancer.

Uptream traffics goes directly from server to client.

Loadbalancing



Loadbalancing

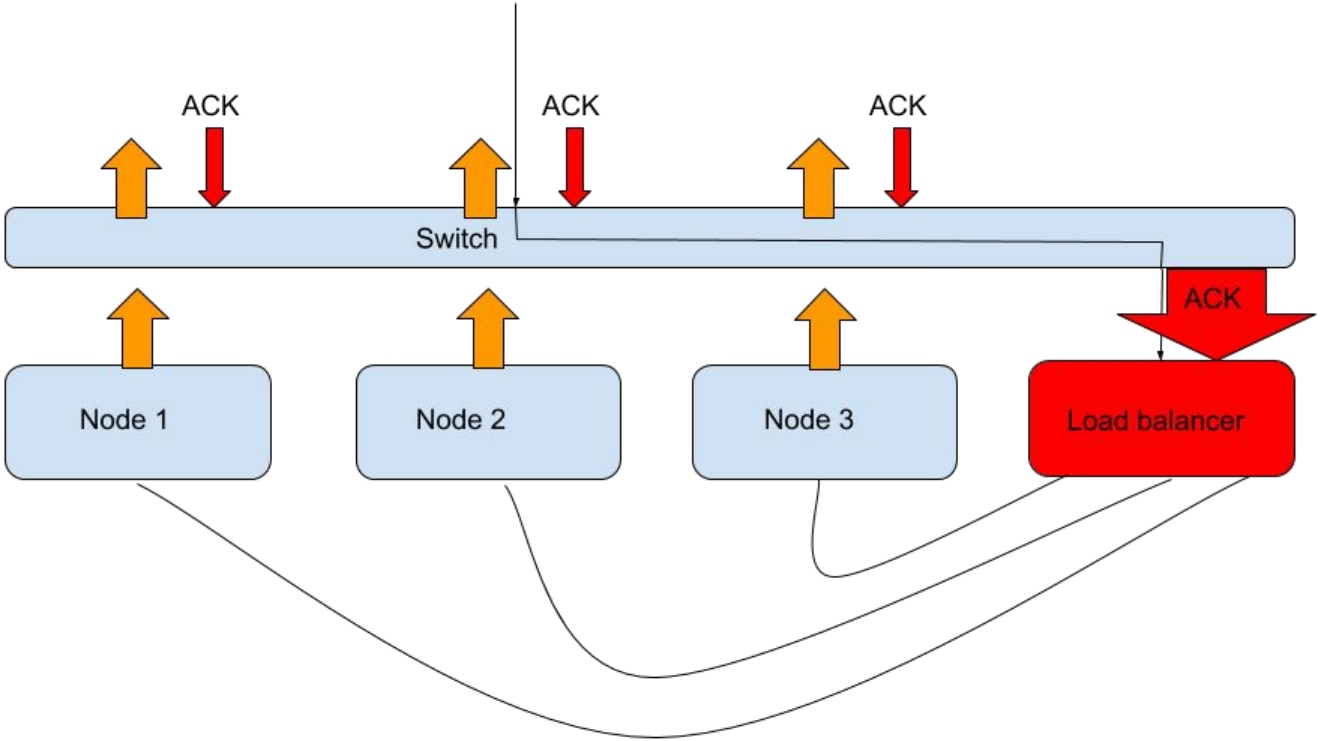
A nice approach is "Direct Routing"; IPVS-DR. A linux kernel module that acts as a loadbalancer.

Only the downstream traffic flows through the loadbalancer.

Uptream traffics goes directly from server to client.

Problem: for large volumes the loadbalancer still drowns in the downstream traffic rates (200.000 pps (mostly ACKs))

Loadbalancing



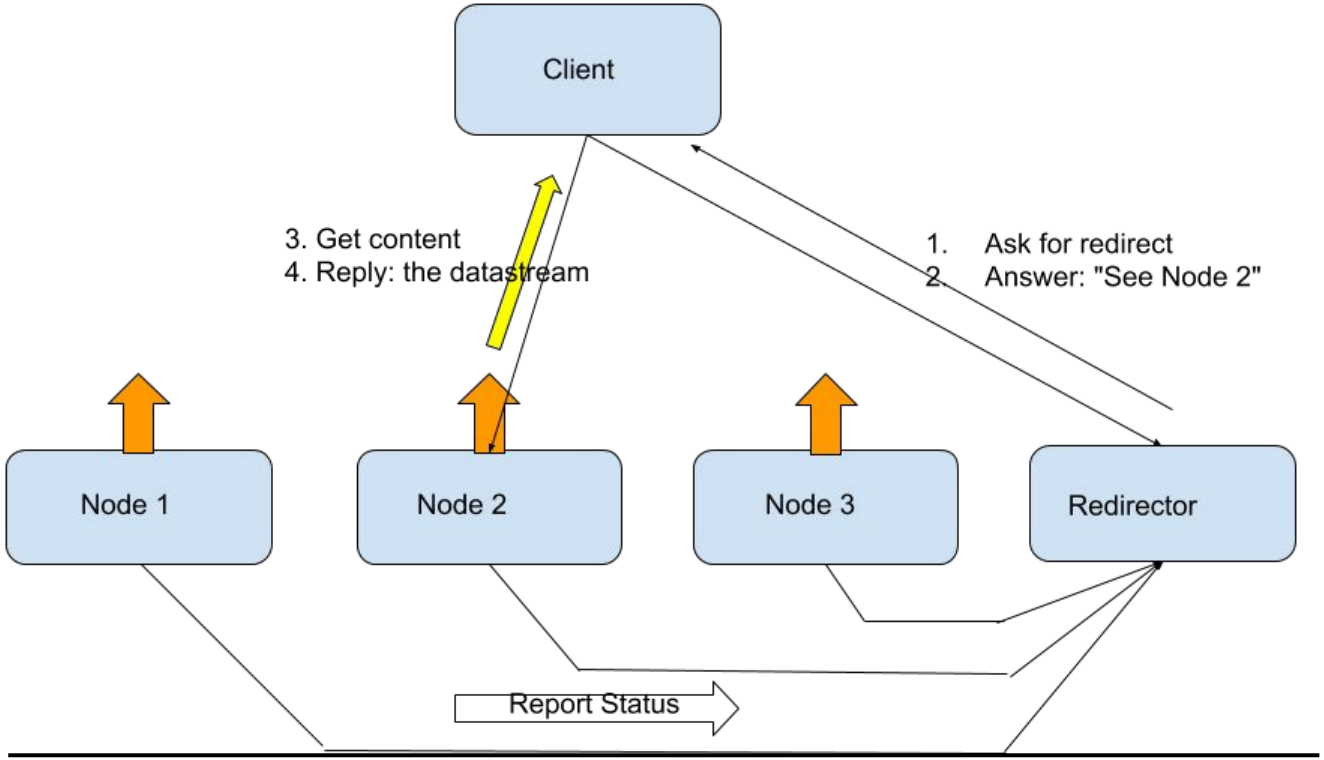
Loadbalancing

Our solution: PMLB "Poor Mans Load Balancing"

Not a loadbalancer but a redirector. Can be written in programming language of choice.

- Request comes in at the redirector
- The redirector has knowledge about a pool of streaming servers and redirects the client to one of them
- Then, the client has a direct 1-on-1 TCP connection to the chosen streaming server

Loadbalancing



Loadbalancing

This redirector can do more stuff besides loadbalancing:

- generate secured links to the streaming servers
- do GeoIP checks on incoming requests
- do anti-deeplink verification
- generate statistics
- know about up/down status of streaming servers
- route traffic to specific clusters of streaming servers
- generate "stadium is full" redirects

Content protection

Content Protection

Problem: in HLS/HDS/MSS streaming we want to protect both the manifests *and* the data chunks. So only checking at the redirector does not suffice. We have to check each and every request.

Solution: the redirector generates unique redirects to the streaming servers. The URL contains a token in which (a.o.) the IP address of the client is encoded.

Content Protection

If the redirector generates private tokens, the streaming servers have to check these tokens for validity

This is where nginx+lua comes in.

Very easy to run custom lua code in nginx

Very easy to write lua code to do the checking

Token based on shared secret between redirector en streaming server

Content Protection

Generate a secured redirect: (redirector, php)

```
private function maketoken($clip, $ip, $secret)
{
     $auth=join('|', array($clip, $ip, $secret));
     return md5($auth);
}

private function redirect($clip, $scheme, $server, $ip, $secret)
{
     $token = $this->maketoken($clip,$ip);
     return "$scheme://$server/secure/$token/$clip";
}
```

Content Protection

```
local function validate()
     local uri_pattern = "^/secure/([^/]*)([^?]*).*"
     local matches,_,token,clip =
               string.find(ngx.var.request_uri, uri_pattern)
     if (matches) then
               local p = "|"
               local ourtoken =
                    ngx.md5(clip .. p .. ip .. p .. config.secret)
               if (ourtoken ~= token) then
                         ngx.exit(ngx.HTTP_FORBIDDEN)
               end
     else
               ngx.exit(ngx.HTTP_FORBIDDEN);
     end
end
```

Content Protection

- We use Lua, because...
- Lua is a nice, clean, simple to learn, simple to read language.
- Same can be achieved using NGINX javascript module
- Javascript knowledge is more widespread than Lua.

Addditional redirector functionality

Additional functionality

GeoIP protection

Use Maxmind geolocation database + php pecl module

$countrycode = geoip_country_code_by_name($ip);

Additional functionality

Anti deeplink verification

Client generates time-based tokens with shared secret

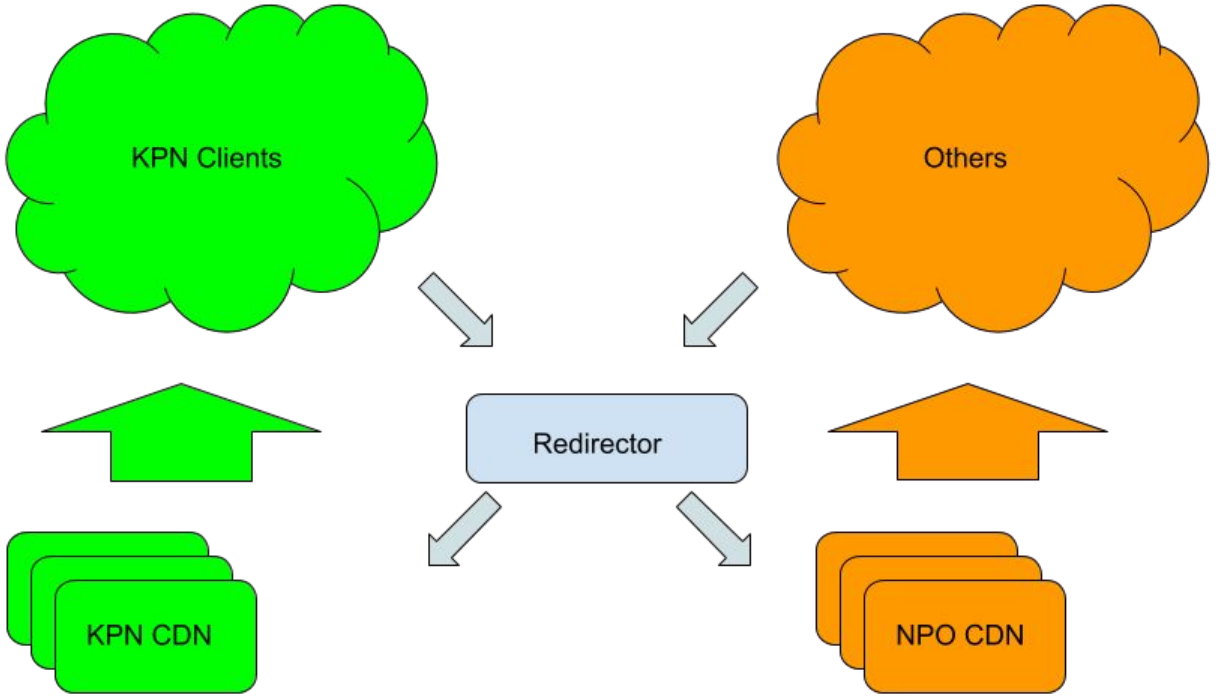
```
$secret = "correct horse battery staple";
$uri = "/clips/protected/for-your-eyes-only.mp4";
$server = "content.omroep.nl";

$t_hex = sprintf("%08x", time());
$token = md5($secret.$uri.$t_hex);
$url = sprintf("http://%s/secure/%s/%s%s",
                                $server,$token,$t_hex,$uri);
```

Additional functionality

route traffic to specific clusters of streaming servers

- Each network has its own AS number
- Maxmind database exists to lookup AS numbers by IP.
- Redirector recognizes specific AS numbers and redirects clients to a cluster specific to that AS number

Additional functionality



SSL Optimizations

SSL Optimizations

The challenge:

- Most websites use https nowadays
- Streaming used to be a separate protocol (e.g. rtsp), but nowadays (HLS,HDS, MSS) it's http(s)
- Your browser sees that as part of the website
- So it complains about mixed content if streaming is done over http instead of https
- Net result: streaming servers *have* to do https

SSL Optimizations

SSL is very CPU intensive

Streaming is usually network bound. With SSL switched on that shifts to CPU bound.

NGINX is already great for running doing SSL, but there are some tweaks to increase efficiency

SSL Optimizations

Netflix has the best solution:

- push SSL handling to the kernel
- now network traffic is again zero copy. I.e. the outgoing data needs to be touched only once and not multiple times for encryption
- works on FreeBSD

But… we use Linux.

SSL Optimizations

Linux optimizations

- run a new kernel!
- lots of info on the internet
- smp_affinity
- receive packet steering (on old hardware)
- codel

SSL Optimizations

Openssl optimizations

- add support for chacha-poly ciphers

NGINX optimizations

- pcre-jit

Bonus: /server-status in Lua

/server-status in Lua

Commercial variant has a nice status (ngx_http_api_module)

Open Source nginx server-status is *very* terse:

```
Active connections: 523
server accepts handled requests
 2217414 2217414 4736545
Reading: 0 Writing: 234 Waiting: 289
```

Can we do better using a bit of lua code?

/server-status in Lua

Sure!

Add a hook for when the request starts and when it's done:

```
rewrite_by_lua_block { s = require "stats"; s.incoming() }

log_by_lua_block { s = require "stats"; s.outgoing() }
```

/server-status in Lua

Use dicts to keep persistent state:

```
local log_dict = ngx.shared.log_dict

function m.outgoing()
    ...
    incr(log_dict, "bytes_sent", ngx.var.bytes_sent)
    incr(log_dict, "total_reqs", 1)
    ...
end
```

Add any other logic you wish

/server-status in Lua

Add a hook to display the counters:

```
location /server-stats
       {
              content_by_lua_block { s = require "stats"; s.showstatus() }
              include       /e/lib/ip/omroepnet.nginx;
              include       /e/lib/ip/beheer.nginx;
              deny all;
       }
```

/server-status in Lua

Display the default counters

```
local function stub_status(total_reqs)
       -- Active connections: 6112
       -- server accepts handled requests
       --  224956831 224956831 614701793
       -- Reading: 0 Writing: 14 Waiting: 6096
       local total_reqs = log_dict:get("total_reqs") or 0
       ngx.say("Active connections: ", ngx.var.connections_active)
       ngx.say("server accepts handled requests")
       ngx.say(0, " ", 0, " ", total_reqs)
       ngx.say("Reading: ", ngx.var.connections_reading,
              " Writing: ", ngx.var.connections_writing,
              " Waiting: ", ngx.var.connections_waiting)
end
```

/server-status in Lua

Display the custom counters

```
local function extended_status()
        ngx.say("")
        ngx.say("time ", os.date("%Y%m%d %H:%M:%S", now))
        ngx.say("server_pid ", ngx.var.pid)
        ngx.say("lua_version ", _VERSION)
        ngx.say("nginx_version ", ngx.var.nginx_version)
        ngx.say("bytes_sent ", log_dict:get("bytes_sent") or 0, " ", normalize("%6.2f%s",
log_dict:get("bytes_sent") or 0))
        ngx.say("http_reqs ", log_dict:get("http_reqs") or 0)
        ngx.say("https_reqs ", log_dict:get("https_reqs") or 0)
end
```

/server-status in Lua

Example results

```
Active connections: 528
server accepts handled requests
0 0 4735057
Reading: 0 Writing: 252 Waiting: 275

time 20180924 10:37:09
server_pid 5103
lua_version Lua 5.1
nginx_version 1.15.3
bytes_sent 15247571421763  13.87T
http_reqs 475262
https_reqs 4259795

Sliding averages:
counter           five_sec        /s   one_min        /s   one_day        /s
bytes_sent         271.21M  54.24M/s   877.74M  14.63M/s     1.30T  15.78M/s
total_reqs              67   13.40/s       232    3.87/s    430169    4.98/s
http_reqs                5    1.00/s        69    1.15/s     35566    0.41/s
https_reqs              11    2.20/s       333    5.55/s    394574    4.57/s
```

Questions?